University of Pennsylvania
Department of Electrical and Systems Engineering
Electronic Design Automation

ESE535, Spring 2015 — Assignment # 5 Exercise — Wednesday, February 11

Due: Assign 5: Thursday, February 19, 10PM

This is just the exercise supplement to assignment 5. It is worth 15 of 100 points for assignment 5. See the Assignment 3–6 for the project portion.

Exercise:

Consider the graph in Figure 1. For parts 1–4, asume this is scheduled onto a VLIW datapath with 5 functional units, each of which takes one cycle to perform an operation:

1. What is the latency bound? (critical path bound in class)
2. What is the compute bound? (resource bound in class)
3. Show the schedule that results from using the algorithm shown in Figure 2 to schedule the graph ($F = 5$, $T_{op} = 1$). (*N.b.* This algorithm is not necessarily good; in fact we have seen better in class. This is a setup for your comparison in the following part.)
4. Identify a better schedule for the graph (perhaps using an algorithm from class). Compare the makespan of the schedules. Explain how you found this better schedule (*e.g.* what algorithm did you use? what observations about the graph allowed you to find a better schedule?).
5. Provide good schedules (best you can find) and identify the resource bound and makespan for VLIW datapaths with 4, 3, 2, 1 and units (still one cycle per operation).
6. Plot makespan versus the number of functional units for the data you collected above (parts 4 and 5).

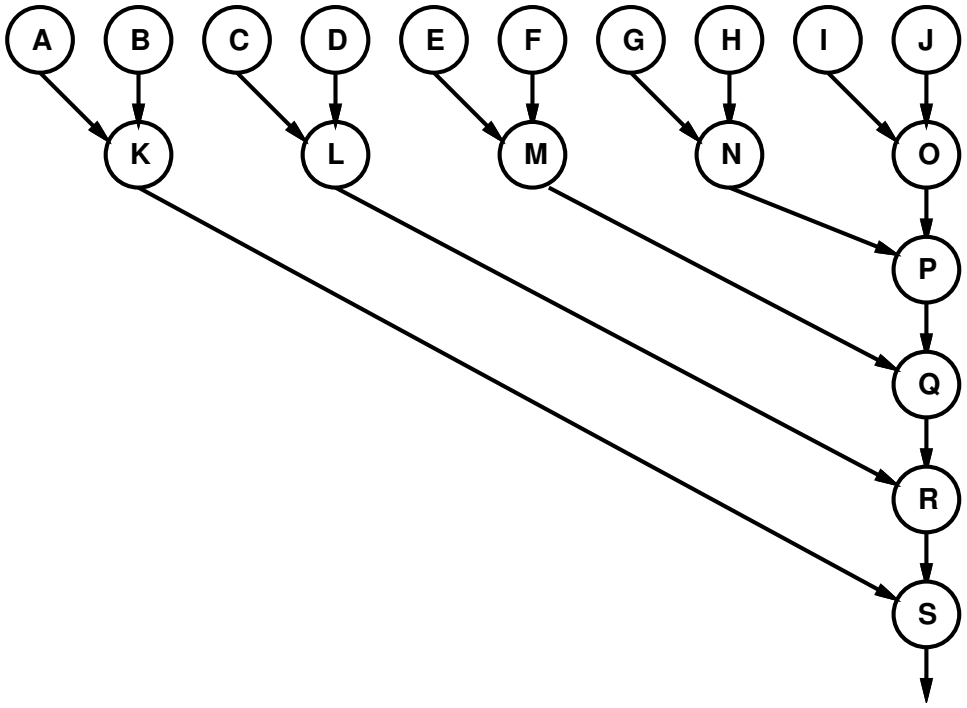

Figure 1: Sample Graph to Schedule

```
Input: Directed Dataflow Graph G = (V, E),
       Number of functional units F,
       Cycles per Operation T_op

//Unscheduled is a set that holds all unscheduled nodes.
Unscheduled=Empty Set
// ReadyQueue holds nodes that are ready to execute.
// Data extracted from Queue is in strict First-In-First-Out (FIFO) order.
ReadyQueue=Empty Queue
// Cycle is an integer indicating the cycle we are scheduling
Cycle=0
// Mark all nodes as unscheduled.
for each v_i ∈ V
    Unscheduled.Add(v_i)
    // Initialize the ReadyQueue with nodes which depend on nothing.
    if v_i has no predecessors in G
        ReadyQueue.Add(v_i)
// Assign nodes to functional units and time slots.
While (not(Empty(Unscheduled)))
    fu=0; // fu is a counter to keep track of assigned functional units
    Running=Empty Set // nodes scheduled at Cycle
    // Start as many nodes running as possible.
    while ((fu< F) and (not(Empty(ReadyQueue))))
        tmpNode=ReadyQueue.removeOldest()
        Assign tmpNode to Functional Unit fu at cycle Cycle (fu++)
        Running.add(tmpNode)
    // Advance Cycle to completion time of these jobs.
    Cycle=Cycle+T_op
    // Mark nodes as schedule.
    for each node n_i ∈ Running
        Unscheduled.Remove(n_i)
    // Put all nodes enabled by the nodes which just completed into the ReadyQueue
    for each node n_i ∈ Running
        for each successor s_i to node n_i
        if no predecesors to s_i ∈ Unscheduled
            ReadyQueue.Add(s_i)
```

Figure 2: Simple First-Come-First-Served Scheduling Algorithm